A PARALLEL ALGORITHM FOR CHANNEL ROUTING ON A HYPERCUBE

Randall Brouwer and Prithviraj Banerjee

Computer Systems Group
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1101 W. Springfield Ave.
Urbana, IL 61801
(217) 333-6564

ABSTRACT

In this paper we present a new parallel simulated annealing algorithm for channel routing on a P processor hypercube. The basic idea used is to partition a set of tracks equally among processors in the hypercube. In parallel, $P/2$ pairs of processors perform displacements and exchanges of nets between tracks, compute the changes in cost functions, and accept moves using a parallel annealing criteria. Through the use of a unique distributed data structure, we are able to minimize message traffic and add versatility and efficiency in a parallel routing tool. The algorithm has been implemented and is being tested on some of the popular channel problems from the literature.

Acknowledgement: This research was supported in part by the National Aeronautics and Space Administration under contract NASA NAG 1-613. Please address all correspondence to second author.

1. Introduction

Over the past few years, much research has been directed toward ways to apply simulated annealing, a multivariate optimization technique [1], to many difficult problems in computer aided design. Some of these include logic minimization [2], cell placement [3,4], global routing [5], and detailed routing [6]. These research efforts have demonstrated that near-optimal results can be achieved for NP-hard problems using simulated annealing. The major drawback to the use of simulated annealing is the excessively long run times required to achieve good results. Some recent work has applied simulated annealing to parallel architectures to reduce the long run times. Extremely good results have been shown for parallel standard cell placement algorithms [7,8,9,10] and parallel global routing algorithms [11].

The problem of channel routing deals with a rectangular wiring area called a channel with pins on the top and bottom edges of the channel, and a collection of nets which are sets of pins that must be interconnected. Nets are routed with horizontal wire segments on one layer and vertical wire segments on another. Connections between the two layers are made through *via* holes. The objective of a channel router is to interconnect all the nets so as to minimize various criteria such as the area of the channel and the number of vias used. Several algorithms for channel routing exist in the literature that apply heuristics. Since they are greedy algorithms, they have the possibility of getting stuck at local minima [12,13,14,15,16].

Recently Leong, Wong, and Liu [6] have proposed a uniprocessor simulated annealing channel routing algorithm; however, long run times are required to achieve good routings. The algorithm they employed requires the detection of cycles in the vertical constraint graph; however, it is not feasible to detect cycles in a parallel processing environment. Furthermore, their algorithm is unable to handle switch-box routing, obstacle avoidance, and unrestricted doglegging, all of which are important in any good routing tool.

In this paper, we present a new parallel simulated annealing algorithm for channel routing. Parallelizing a uniprocessor algorithm should provide faster run times. Additionally, we hope to achieve better convergence due to the parallel state changes as was experimentally observed in a parallel algorithm for cell placement [10]. The algorithm we propose is also more versatile, as it can easily be extended to include switch-box routing, unrestricted doglegging, and obstacle avoidance. Unlike the simulated annealing channel router mentioned earlier, our algorithm permits overlap between distinct nets in early stages of the annealing process, allowing more freedom for getting out of local minima and finding a global minimum.

2. Description

2.1. Parallel Architecture

The algorithm we present in this paper is targeted for implementation on the Intel iPSC Hypercube Computer. A hypercube computer is a message passing architecture consisting of $P = 2^d$ processor nodes in which each node is directly connected to d other nodes. Communication between the processor nodes is restricted to passing variable sized messages between adjacent nodes. Figure 1 shows a three-dimensional hypercube.

2.2. Channel Partitioning and Processor Mapping

To partition the data uniformly among all processors of the hypercube, adjacent tracks of the channel are grouped together and assigned to a single processor. We define the processor domain PD_i as the set of adjacent tracks along with all nets currently assigned to the tracks of the channel over which processor P_i is given control. Consecutive domains are assigned to adjacent processor nodes in the hypercube topology to provide a balance in communication distance. Figure 2 shows a channel partitioning for a hypercube with dimension d=3. This partitioning arrangement allows for net displacement to adjacent domains and to other domains separated by large vertical distances

in the channel. At high temperatures in the annealing process, the algorithm can allow moves across both small and large distances in the hypercube. At lower temperatures, moves along dimensions that correspond to large vertical distances are inhibited.

2.3. Moves

In the restricted doglegging version currently implemented, four move types are provided for permuting the current state of the channel into a new state. Given that P_i and P_j are connected in dimension k of the hypercube, the moves are as follows:

Move 1: P_i and P_j independently displace a net from one track in their respective domains to another track in the same domain.

Move 2: P_i displaces net from track in PD_i to PD_j of P_j.

Move 3: P_i and P_j independently exchange track assignments of two nets in their respective domains.

Move 4: P_i exchanges net from PD_i with net from PD_j of P_j.

Sequences of these exchange and displace moves are sufficient for all permutations of the channel state. These moves are chosen randomly, with relative frequencies of 4:4:1:1 respectively.

2.4. Algorithm

The annealing algorithm we use is outlined in Figure 3. The value of inner_loop_count is specified to be 100 times the number of nets in the given channel routing problem. During each iteration of the inner loop, each of the d dimensions of the hypercube is sequenced through in which each of the $P/2$ processor pairs in dimension k attempt one of the four types of moves in parallel.

2.5. Annealing Schedule

The annealing temperature is adjusted based on the following schedule:

$$T_{NEW} = ALPHA(T) \times T_{OLD}$$

in which the function $ALPHA(T)$ ranges from 0.8 for large values of T to 0.95 for small values of T. This schedule allows more permutations at low annealing temperatures to make many small improvements.

To determine the initial temperature, 100 random moves with a positive cost change are evaluated without accepting any of them. The average cost change $\Delta COST_{AVE}$ for those moves is then calculated, and we solve for T_{INIT} as follows:

$$T_{INIT} = -\frac{\Delta COST_{AVE}}{ln(0.8)}$$

2.6. Cost Function

The cost for a given state of the channel is a function of the amount of overlap between unique nets(OL), the length of the nets(NL), the width of the channel(WC), and the fraction of the track not occupied by nets(FU). For each move, the cost change incurred if the move was accepted is calculated as follows and used to determine move acceptance.

$$\Delta COST = \alpha \times (\Delta OL) + \beta \times (\Delta NL) + \gamma \times (\Delta CW) + \delta \times (\Delta FU)$$

Since move costs are calculated in parallel, the calculated cost change is only an estimate because it does not account for interactions on the channel state by other processor pairs accepting moves. Jones and Banerjee [10] have shown, experimentally, that this property of parallel simulated annealing improves the overall convergence for the cell placement problem. We are expecting to see the same benefits in the channel router problem.

2.7. Distributed Data Structure

Since a hypercube computer is a message passing local memory parallel architecture, there is no shared memory, and one cannot assume the use of a central data structure for storing all of the channel state information. We therefore propose a distributed data structure among processors in the hypercube such that each processor only stores the information that it needs for performing its computations. The data structures we propose help minimize the amount of message passing required, reduce the memory space used for storing the necessary data, and take advantage of the fact that the cost of a message is almost independent of the message size. For each net n in PD_i of p_i the positions of the horizontal and vertical segments of net n, along with the positions of all other vertical segments of nets also occupying the columns of net n must be stored. All of this data is necessary for calculating the expected overlap, channel width, and net length changes for a given move. The data structures used for storing the track and column data for one processor node is shown in Figure 4.

2.8. Net Location Updating

To ensure straightforward and accurate updating of net positions in the new channel state, the position data for those nets is passed from node to node along a Hamiltonian cycle through every node of the hypercube. A Hamiltonian cycle in a graph is defined as a cycle in a graph which traverses every node of the graph exactly once. A Hamiltonian cycle in a 3-dimensional hypercube is shown in bold lines in Figure 1. Each node updates the data it has and then forwards the message to the next node along the cycle. All updating completes within P time steps.

3. Implementation

We have implemented the above algorithm using 3500 lines of C code using an Intel Hypercube Simulator(Version 3.0) running on a Sun 3/50 workstation operating under Sun Unix 4.2. The initial version of our program was debugged one week ago. We are presently carrying out tests

of our parallel algorithm on various test cases. Figure 5 shows an example solution of a problem found in the literature [13].

Figure 6 shows a plot of the annealing channel cost as a function of temperature. We will be reporting the results of our algorithm for many of the other conventional channel routing test cases in the final paper at the conference. and we plan to implement this version on an actual hypercube and report on the performance (ie. speedup, etc.) at the conference.

4. Conclusions

In this paper we have proposed a new parallel algorithm for simulated annealing channel routing for implementation on a hypercube computer. By the use of a novel distributed data structure and partitioning of the channel, we have a versatile algorithm for channel routing that is easily extensible to switch-box routing and obstacle avoidance routing.

REFERENCES

[1] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, pp. 671-680, May 1983.

[2] J. Lam and J. M. Delosme, "Logic Minimization Using Simulated Annealing," *Proc. IEEE Int. Conf. Computer-Aided Design (ICCAD-86)*, pp. 348-351, Nov. 1986.

[3] C. Sechen and A. S. Vincentelli, "TimberWolf3.2: A New Standard Cell Placement and Global Routing Package," *Proc. 23rd Design Automation Conf.*, pp. 432-439, Jun. 1986.

[4] L. K. Grover, "A New Simulated Annealing Algorithm for Standard Cell Placement," *Proc. Int. Conf. on Computer-Aided Design (ICCAD-86)*, pp. 378-380, Nov. 1986.

[5] M. P. Vecchi and S. Kirkpatrick, "Global Wiring by Simulated Annealing," *IEEE Transactions on Computer-Aided Design*, vol. CAD-2, No. 4, pp. 215-222, October 1983.

[6] H. W. Leong, D. F. Wong, and C. L. Liu, "A Simulated Annealing Channel Router," *Proc. 22nd Design Automation Conf.*, pp. 226-228, June 1985.

[7] A. Casotto, F. Romeo, and A. Sangiovanni-Vincentelli, "A Parallel Simulated Annealing Algorithm for the Placement of Macro-Cells," *Proc. IEEE Int. Conf. Computer-Aided Design (ICCAD-86)*, pp. 30-33, Nov. 1986.

[8] R. A. Rutenbar and S. A. Kravitz, "Layout by Annealing in a Parallel Environment," *Proc. IEEE Int. Conf. on Computer Design (ICCD-86)*, pp. 434-437, Oct. 1986.

[9] P. Banerjee and M. Jones, "A Parallel Simulated Annealing for Standard Cell Placement on a Hypercube Computer," *Proc. IEEE Int. Conf. Computer-Aided Design (ICCAD-86)*, Nov. 1986.

[10] M. Jones and P. Banerjee, "Performance of a Parallel Algorithm for Standard Cell Placement on the Intel Hypercube," *Proc. 24th Design Automation Conf.*, June 1987.

[11] M. J. Chung and K. K. Rao, "Parallel Simulated Annealing for Partitioning and Routing," *Proc. IEEE Int. Conf. on Computer Design (ICCD-86)*, pp. 238-242, Oct. 1986.

[12] A. Hashimoto and J. Stevens, "Wire Routing by Optimizing Channel Assignment," *Proc. 8th Design Automation Conf.*, pp. 214-224, June 1971.

[13] T. Yoshimura and E. S. Kuh, "Efficient Algorithms for Channel Routing," *IEEE Trans. Computer-Aided Design*, vol. CAD-1, pp. 25-25, Jan. 1982.

[14] R. L. Rivest and C. M. Fidducia, "A Greedy Channel Router," *Proc. 19th Design Automation Conf.*, pp. 418-424, June 1982.

[15] D. Deutsch, "A Dogleg Channel Router," *Proc. 13th Design Automation*, pp. 425-433, June 1976.

[16] M. Burstein and R. Pelavin, "Hierarchical Channel Router," *Proc. 20th Design Automation Conf.*, pp. 591-597, June 1983.

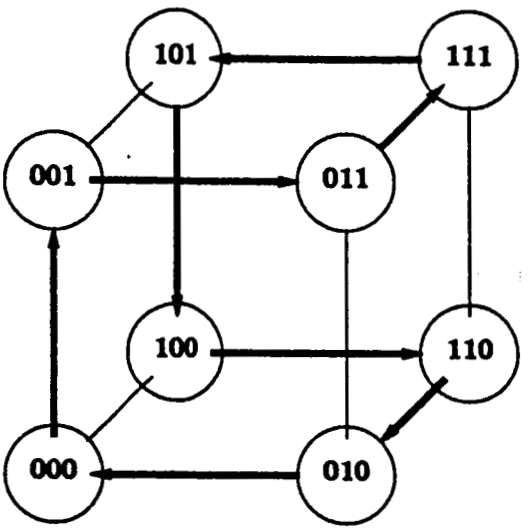


Figure 1. 3-Dimensional Hypercube Showing a Hamiltonian Cycle

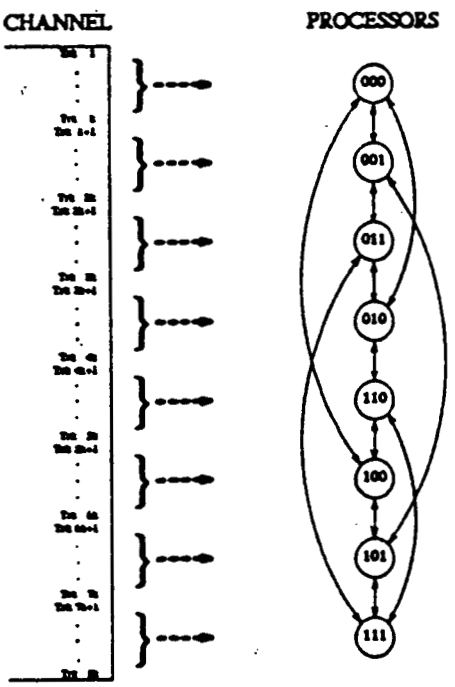
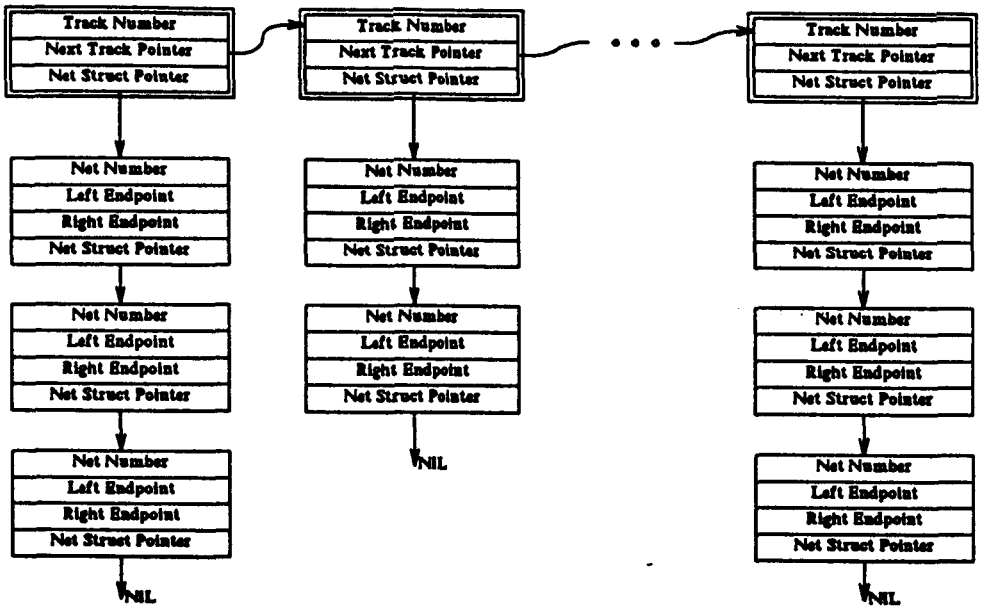


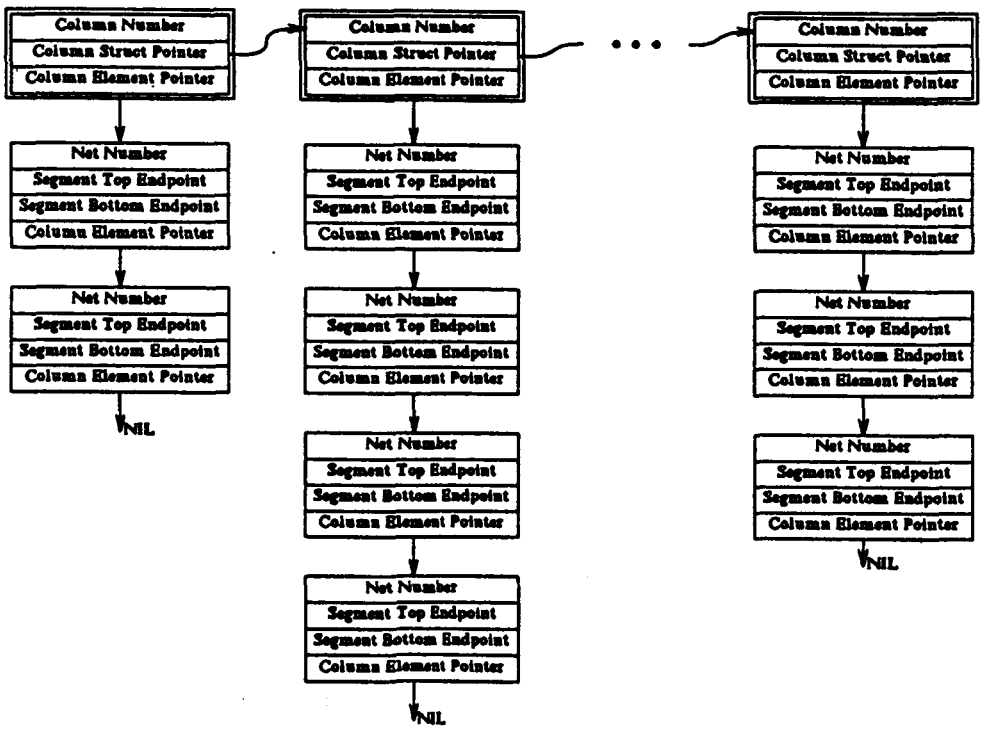
Figure 2. Channel Map onto Hypercube of 3 Dimensions

STEP 1. Perform track assignments to P processors.
STEP 2. Determine initial annealing temperature.
STEP 3. While "Stopping criteria" : temperature $< \epsilon$ not reached
STEP 4. Generate new temperature according to annealing schedule
STEP 5. For inner_loop_count = 1 to USER_PARAMETER
STEP 6. For each dimension k = 0 to log(P)-1 do
STEP 7. Randomly select P/2 moves (exchange or displacement of nets) in parallel among pairs of PEs connected in dimension k.
STEP 8. Evaluate change in cost for each move between pairs of PEs independently.
STEP 9. Accept/reject moves based on exponential function independently.
STEP 10. Broadcast new net locations to all other processors using Hamiltonian cycle.
STEP 11. ENDFOR; ENDFOR; ENDWHILE;

Figure 3. Parallel Algorithm for Channel Routing



(a)



(b)

Figure 4. (a) Track Data Structure (b) Column Data Structure

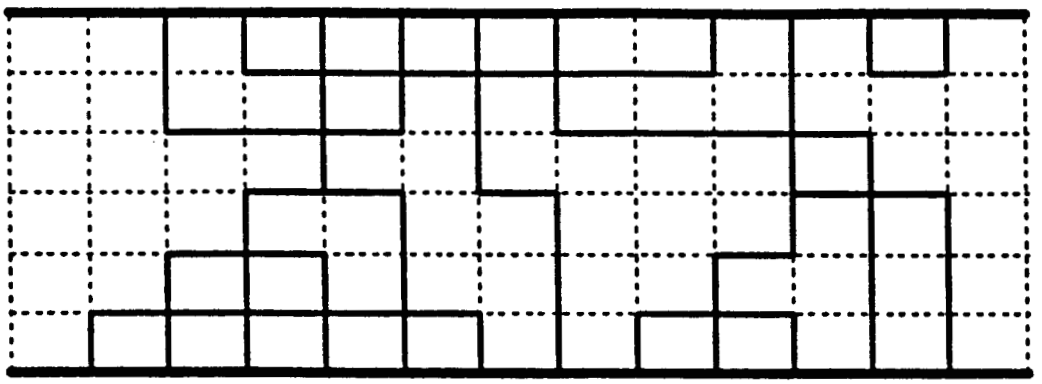


Figure 5. Example Routing Solution

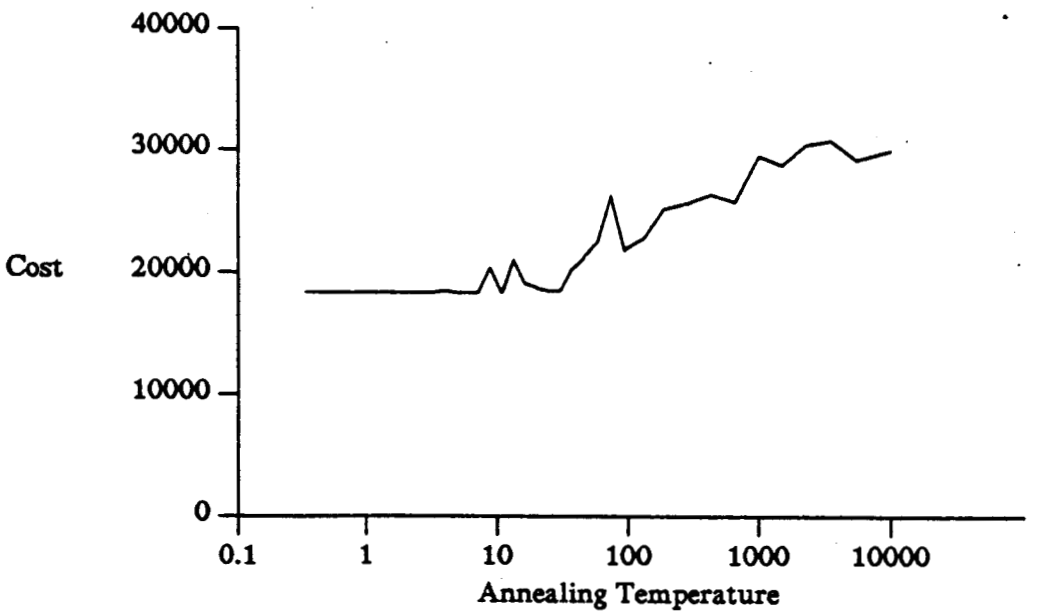


Figure 6. Temperature vs. Cost